Modelling the Shallow Water Flow

Anubhav Singh
<anubhavs@student.unimelb.edu.au>

1 Introduction

The report ellaborates the methods used to solve the Shallow Water Equations(SWEs) and provides a comprehensive analysis of strong and weak scaling behavior of openMP and MPI parallel code(respectively) developed to solve the equations on multi-processor systems/ supercomputers. The sixth order finite difference method was applied for spatial discretization of the equations and Fourth order Runge-Kutta method was used for discretiztion in time domain. The parallelized program was executed on super computer and the output visualized using ParaView.

The shallow water equations consists of the following three first-order linear homogenous partial differential equations(PDEs)

$$\delta v_x/\delta t = -v_x.\delta v_x/\delta x - v_y.\delta v_x/\delta y - g.\delta h/\delta x \quad (1)$$

$$\delta v_y/\delta t = -v_x.\delta v_y/\delta x - v_y.\delta v_y/\delta y - g.\delta h/\delta y \quad (2)$$

$$\delta h/\delta t = -\delta(v_x.h)/\delta x - \delta(v_y.h)/\delta y \quad (3)$$

While the time discretization step is sequential and dependent on the previous outputs, the space discretization is independent. The output of spatial discretization for a particular grid point(x, y) is independent of output of spatial discretization on any other points(at the same time step), which presents an opportunity to perform spatial discretization in parallel. While the computation time for a single grid point is small, there can be millions of grid points in a problem (depending upon parameters chosen), and theoritically parallelization can greatly reduce the computation time. In this project, the c++ code was parallelized using OpenMP and MPI library methods.

2 Methods

2.1 Finite Difference

Finite difference method involves replacing the spatial derivative term with a stencil, which is a difference quotient term appoximately equal to the actual derivative. A sixth-order accurate finite stencil was used for spatial discretization in this project.

2.1.1 Sixth-order accurate finite stencil

Derivation:
We used the generalized matrix form of the system of equation derived from taylor serires expansion in MCEN90031 Lecture Slides 157 − 158. We apply M=3 to derive the sixth order stencil.

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -3 & -3 & -3 & -0 & 3 & 3 & 3 \\ 9/2 & 9/2 & 9/2 & 0 & 9/2 & 9/2 & 9/2 \\ -9/2 & -9/2 & -9/2 & 0 & 9/2 & 9/2 & 9/2 \\ \vdots & & & & & & \end{bmatrix} \begin{bmatrix} a_{-3} \\ a_{-2} \\ a_{-1} \\ a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (4)$$

We get the following values of a_{M_i} upon solving the above system of linear equation.

$$a_{-3} = -1/60,\ a_{-2} = 3/20, a_{-1} = -3/4,$$
$$a_3 = 1/60,\ a_2 = -3/20, a_1 = 3/4$$
$$, a_0 = 0,$$

Application on Shallow Water Equations(SWEs):
Applying the spatial discretization on SWEs, we get the following system of equations for all the

grid points{x,y} in the system.

$$\begin{aligned} dv_x/dt|_{x,y} = (&-v_x|_{x,y} * ((1/60) * v_x|_{x+3,y} - \\ &(1/60) * v_x|_{x-3,y} - (3/20) * v_x|_{x+2,y} \\ &+(3/20) * v_x|_{x-2,y} + (3/4) * v_x|_{x+1,y} \\ &-(3/4) * v_x|_{x-1,y})/(\Delta x) \\ &-v_y|_{x,y} * ((1/60) * v_x|_{x+3,x} \\ &-(1/60) * v_x|_{x-3,y} - (3/20) * v_x|_{x+2,y} \\ &+(3/20) * v_x|_{x-2,y} + (3/4) * v_x|_{x+1,y} \\ &-(3/4) * v_x|_{x-1,y})/(\Delta y)) \\ &-g * ((1/60) * h|_{x+3,y} - (1/60) * h|_{x-3,y} \\ &-(3/20) * h|_{x+2,y} + (3/20) * h|_{x-2,y} \\ &+(3/4) * h|_{x+1,y} - (3/4) * h|_{x-1,y})/(\Delta x); \end{aligned} \tag{5}$$

$$\begin{aligned} dv_y/dt|_{x,y} = (&-v_x|_{x,y} * ((1/60) * v_y|_{x+3,y} - \\ &(1/60) * v_y|_{x-3,y} - (3/20) * v_y|_{x+2,y} \\ &+(3/20) * v_y|_{x-2,y} + (3/4) * v_y|_{x+1,y} \\ &-(3/4) * v_y|_{x-1,y})/(\Delta x) \\ &-v_y|_{x,y} * ((1/60) * v_y|_{x+3,y} \\ &-(1/60) * v_y|_{x-3,y} - (3/20) * v_y|_{x+2,y} \\ &+(3/20) * v_y|_{x-2,y} + (3/4) * v_y|_{x+1,y} \\ &-(3/4) * v_y|_{x-1,y})/(\Delta y)) \\ &-g * ((1/60) * h|_{x+3,y} - (1/60) * h|_{x-3,y} \\ &-(3/20) * h|_{x+2,y} + (3/20) * h|_{x-2,y} \\ &+(3/4) * h|_{x+1,y} - (3/4) * h|_{x-1,y})/(\Delta y); \end{aligned} \tag{6}$$

$$\begin{aligned} dh/dt|_{x,y} = (&-v_x|_{x,y} * ((1/60) * h|_{x+3,y} - \\ &(1/60) * h|_{x-3,y} - (3/20) * h|_{x+2,y} \\ &+(3/20) * h|_{x-2,y} + (3/4) * h|_{x+1,y} \\ &-(3/4) * h|_{x-1,y})/(\Delta x) \\ &-v_y|_{x,y} * ((1/60) * h|_{x+3,y} \\ &-(1/60) * h|_{x-3,y} - (3/20) * h_{x+2,y} \\ &+(3/20) * h|_{x-2,y} + (3/4) * h|_{x+1,y} \\ &-(3/4) * h|_{x-1,y})/(\Delta y)) \\ &-h|_{x,y} * ((1/60) * v_x|_{x+3,y} - (1/60) * v_x|_{x-3,y} \\ &-(3/20) * v_x|_{x+2,y} + (3/20) * v_x|_{x-2,y} \\ &+(3/4) * v_x|_{x+1,y} - (3/4) * v_x|_{x-1,y})/(\Delta x) \\ &+(1/60) * v_y|_{x+3,y} - (1/60) * v_y|_{x-3,y} \\ &-(3/20) * v_y|_{x+2,y} + (3/20) * v_y|_{x-2,y} \\ &+(3/4) * v_y|_{x+1,y} - (3/4) * v_y|_{x-1,y})/(\Delta y)); \end{aligned} \tag{7}$$

2.2 Fourth-order Runge-Kutta Method

The Runge-kutta method aims to provide better approximate solution to an ODE in comparison to explicit Euler's method. The following equation represent the general form of Runge-kutta method.

$$\phi^{l+1} = \phi^l + \Delta t.g(\phi^l, t, \Delta t) \tag{8}$$

Where g is defined as,

$$g = a_1k_1 + a_2k_2 + \therefore + a_Nk_N \tag{9}$$

and k_n is defined as,

$$k_n = f\{\phi^l + q_{n-1,1}, k_1\Delta t + q_{n-1,2} + \therefore, t^l + p_{n-1}\Delta t\}; \tag{10}$$

Solving for $N = 4$, fourth order runge kutta equation, we get g as,

$$g = (1/6)k_1 + (1/3)k_2 + (1/3)k_3 + (1/6)k_4 \tag{11}$$

and k_n as,

$$k_1 = f\{\phi^l, t^l\}; \tag{12}$$

$$k_2 = f\{\phi^l + (\Delta t/2)k_1, t^l + (\Delta t/2)\}; \tag{13}$$

$$k_3 = f\{\phi^l + (\Delta t/2)k_2, t^l + (\Delta t/2)\}; \tag{14}$$

$$k_4 = f\{\phi^l + (\Delta t)k_3, t^l + (\Delta t/2)\}; \tag{15}$$

The fourth-order Runge-Kutta method was then applied on the semi-discretized system of ODEs derived from spatial discretization using finite difference methods.

2.2.1 Stability Analysis on model ODE

We used model ODE equation below to perform the stability analysis of the fourth-order runge kutta method.

$$d\phi/dt = \lambda\phi \tag{16}$$

Applying Runge-Kutta method to it we get,

$$\begin{aligned} \phi^{l+1} = \phi^l(1 + \lambda\Delta t + (\lambda\Delta t)^2/2 \\ +(\lambda\Delta t)^3/6 + (\lambda\Delta t)^4/24) \end{aligned} \tag{17}$$

$$\begin{aligned} \phi^{l+1} = \phi^0(1 + \lambda\Delta t + (\lambda\Delta t)^2/2 \\ +(\lambda\Delta t)^3/6 + (\lambda\Delta t)^4/24)^l \end{aligned}$$

$$\phi^{l+1} = \phi^0\sigma^l \tag{18}$$

For the method to be surely stable, $\sigma < 1$. *Figure. 1* represents the graphical plot of the stability region.

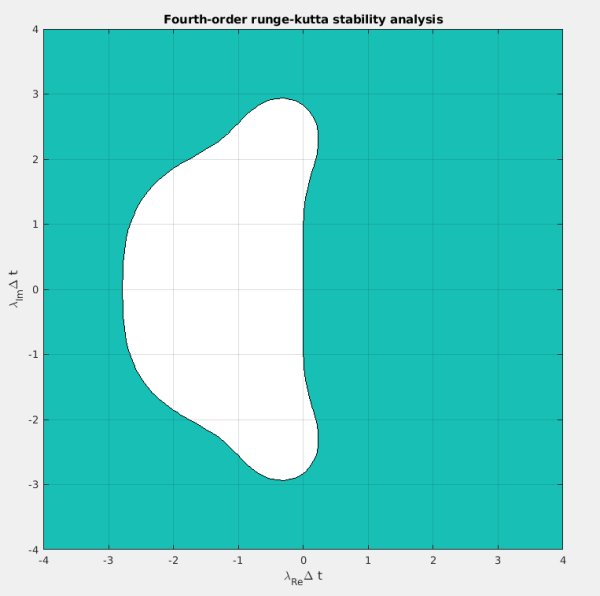

Figure 1: Stability Analysis-fourth order runge-kutta

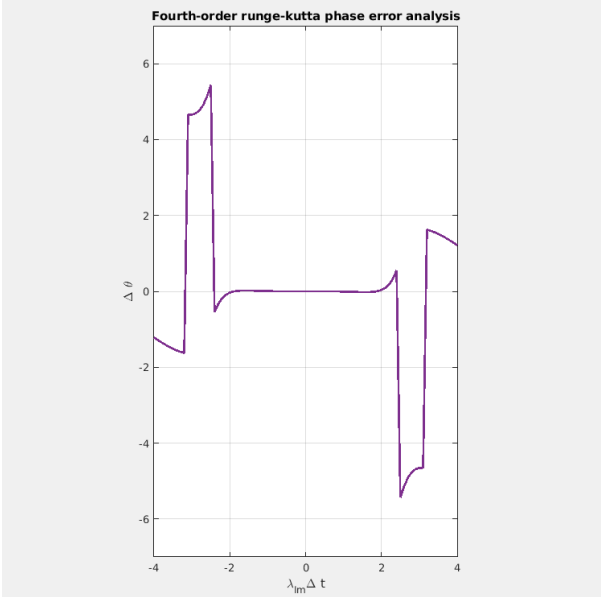

Figure 2: Phase Error-fourth order runge-kutta

To perform phase and amplitude error analysis, λ is assumed to be imaginary in the model ODE, for which the exact solution is: $\phi(t = l\Delta t) = \phi^0 e^{i\lambda_{im} l\Delta t}$.

Using equation 18, we get:

$$\phi^{l+1} = \phi^0 \sigma^l = \phi^0 Z^l e^{il\theta}$$

Now, dividing the two equations, we get:

$$\phi^{l+1} = \phi^0 Z^l e^{il(\theta - \lambda_{im}\Delta t)} \tag{19}$$

Here, phase error is defined as $l(\theta - \lambda_{im}\Delta t)$ and Z is defined as amplitude error.

The *Figure. 2* shows the plot of Phase Error against $\lambda_{im}\Delta t$.

The *Figure. 3* shows the plot of Amplitude Error against $\lambda_{im}\Delta t$.

It can be observed that the phase and amplitude errors magnify enormously outside the stability region showed in *Figure. 1*

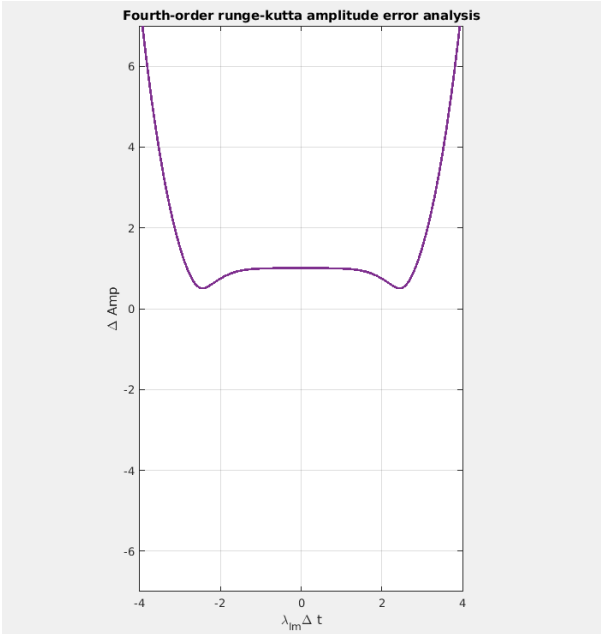

Figure 3: Amplitude Error-fourth order runge-kutta

2.3 Parallelization

2.3.1 OpenMP

The OpenMP interface allows for parallel execution of code on mutiple threads in a shared memory model. OpenMP provides various preprocessor directives allowing for different modes

of parallel execution. An important point about OpenMP is that it can only be used on shared memory systems and in case of distributed memory system is limited to one node only.

In this project, the OpenMP code was developed to analyze strong scaling, by increasing the number of processor and keeping the total problem size to be the same. The *pragma directive* "#pragma omp for" was used on all the loop being used to calculate the spatial discretization for each grid point.

2.3.2 MPI- Message Passing Interface

The MPI library on the other hand exceutes multiple parallel components on seperate processes which don't share the same memory. There are both pros and cons to this approach. While the program can now be run on distributed memory systems, there is overhead of interprocesses communication. Also, part of allocated memory which could be shared in case of shared-memory system, now needs to be duplicated for each process. Hence, requiring more memory over all.

3 Results

3.1 Graphical Output

Following are the graphical output from each of the code developed during the project.

3.1.1 Matlab Code

The parameter chosen were $\Delta x = 1, \Delta y = 1, \Delta t = 0.01$. $\Delta x = \Delta y$ was chosen to be large, to reduce the grid points, because of limited computational power of the local machine running matlab.

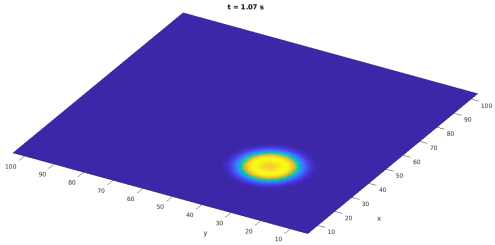

Figure 4: Matlab Plot at t = 0 seconds

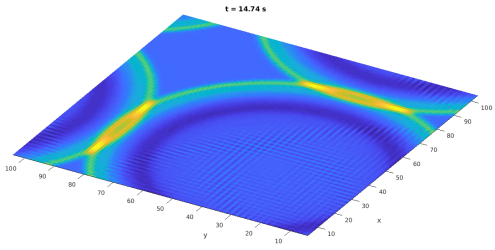

Figure 5: Matlab Plot at t = 15 seconds

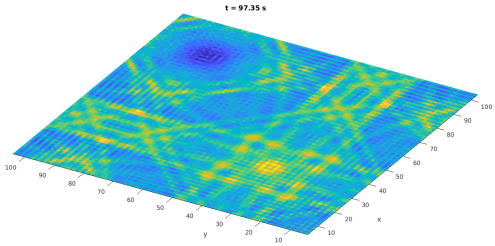

Figure 6: Matlab Plot at t = 97 seconds

3.1.2 C++ Code

The parameter chosen were $\Delta x = 1, \Delta y = 1, \Delta t = 0.01$. $\Delta x = \Delta y$ was chosen to be large because of limited computational power of the local machine running cpp code.

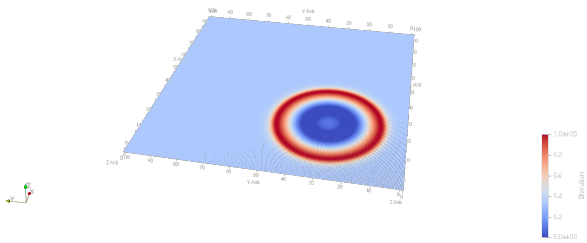

Figure 7: Cpp Plot at t = 5 seconds

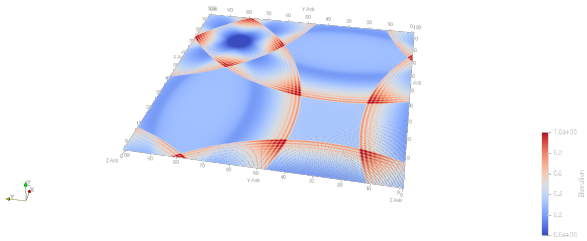

Figure 8: Cpp Plot at t = 25 seconds

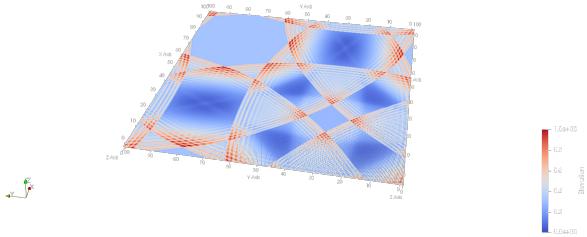

Figure 9: Cpp Plot at t = 40 seconds

3.1.3 OpenMP Code

The parameter chosen were $\Delta x = 0.1, \Delta y = 0.1, \Delta t = 0.01$. $\Delta x = \Delta y$ was chosen to be small because of much higher computational power of the barcoo and snowy supercomputers.

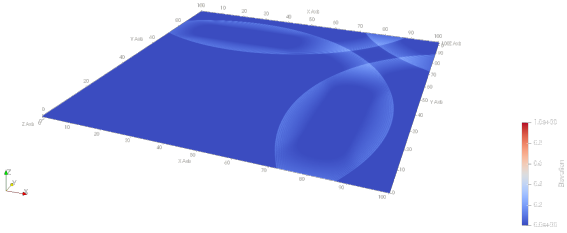

Figure 10: OpenMP Plot at t = 18 seconds

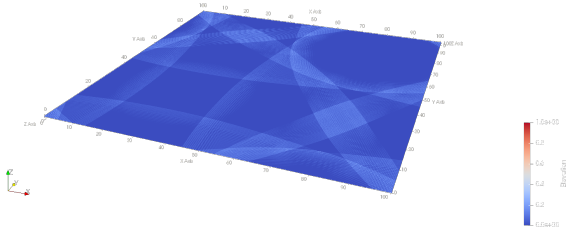

Figure 11: OpenMP Plot at t = 56 seconds

3.1.4 MPI Code

The parameter chosen were $\Delta x = 0.1, \Delta y = 0.1, \Delta t = 0.01$. $\Delta x = \Delta y$ was chosen to be small because of much higher computational power of the barcoo and snowy supercomputers.

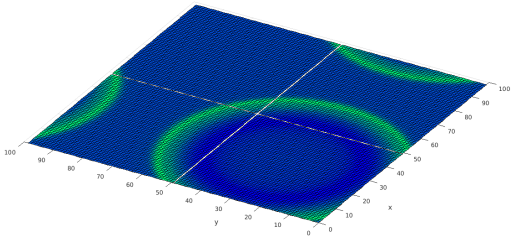

Figure 12: MPI Plot at t = 10 seconds

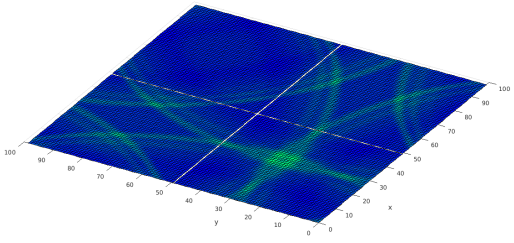

Figure 13: MPI Plot at t = 30 seconds

3.2 Strong-scaling using OpenMP

The OpenMP code was run with varying nummber of processors and fixed problem size. The parameter chosen were $\Delta x = 0.1, \Delta y = 0.1, \Delta t = 0.01$. The file read write operations were removed since they attributed to performance bottleneck.

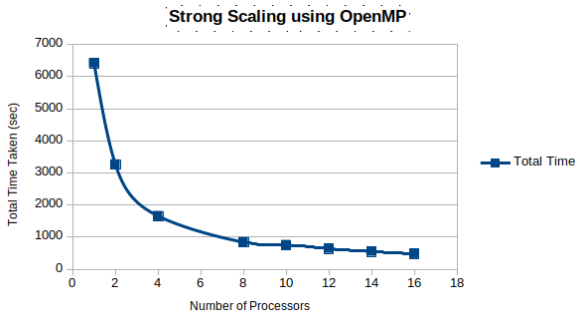

Figure 14: Strong Scaling Analysis using OpenMP

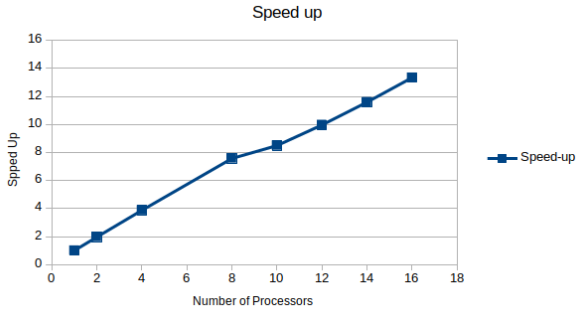

Figure 15: Speed Up: Strong Scaling Analysis using OpenMP

3.3 Weak-scaling using MPI

To perform weak analysis of the MPI code, it was run with varying number of processor and the total problem size was modified in a manner that per processor problem size remains the same. This was acheived by modifying the values of $\Delta x = \Delta y$ according to number of available processors, ensuring that every allocated processor get equal amount of work.

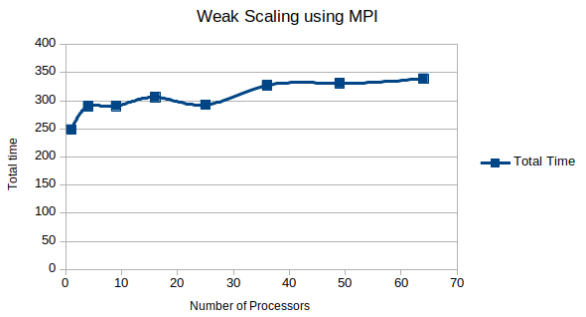

Figure 16: Weak Scaling Analysis using MPI(Time-seconds)

4 Discussion

Various observation were made during the execution of the code on different platforms with different parameters.

1. It was observed that increasing the Δt value in the program resulted in increased variance in output values and on sufficiently high values the explosion of values was evidenced. This could be explained based on the stability condition of Runge-Kutta which requires the Δt to be low.

2. The execution time increased significantly if the output was written to file at each time step. This can be attributed to lower read-write speed on disk compared to physical memory and cache. To reduce the bottleneck, the output file was generated every second, instead of every time-step.

3. Strong Scaling: The results were as expected from Amdahl's law and the time taken was observed to be inversely proportional to number of processors. When the code for file output was included in the program, the execution time increased significantly across all samples, which can be explained considering the high read-write time for disks in comparison to memory and cache. Overall, it was seen that in case if the majority of the code execution-time is dependent on a portion that cannot be parallelized, the performance benefit was insignificant.

4. Weak Scaling Few deviations were observed during the analysis of weak scaling. In general, there was a linear increase in execution time of the code. However, the execution time would sometimes suddenly drop instead of increasing, upon increasing the number of processors. On further experimentation, it was ob-

served that the results were better if the the allocated processors were on the same node on the super-computer. This may be because of reduced interprocess communication delay on the hardware. Apart from that, the execution time increased upon increasing the number of processors, which can be attributed to increased requirement for message passing with more number of processes.